Leetcode(python版) 排序中等题

56合并区间

题目：给出一个区间的集合，请合并所有重叠的区间。

**示例 1:**

**输入:** [[1,3],[2,6],[8,10],[15,18]]

**输出:** [[1,6],[8,10],[15,18]]

**解释:** 区间 [1,3] 和 [2,6] 重叠, 将它们合并为 [1,6].

**示例 2:**

**输入:** [[1,4],[4,5]]

**输出:** [[1,5]]

**解释:** 区间 [1,4] 和 [4,5] 可被视为重叠区间。

思路：

1）先将目标区间数组按X轴从小到大排序。例如：[2,3] [1,2] [6,9] ->[1,2] [2,3] [6,9]

2）扫描排序后的目标区间数组，将这些区间合并成若干个互不相交的区间。例如 [2,3] [1,2] [6,9] ->[1,3] [6,9]

这里分三种情况：

[1,3] [2,6] ->[1,6] 第一个区间的end大于等于第二个区间的start,同时第二个区间的end大于第一个区间的end

[2,8] [3,5] ->[2,8] 第一个区间的end大于等于第二个区间的start,同时第二个区间的end小于第一个区间的end

[1,2] [3,4] -> [1,2] [3,4] 第一个区间的end小于第二个区间的start

代码：

1. # Definition for an interval.
2. # class Interval(object):
3. #     def \_\_init\_\_(self, s=0, e=0):
4. #         self.start = s
5. #         self.end = e
7. **class** Solution(object):
8. **def** merge(self, intervals):
9. """
10. :type intervals: List[Interval]
11. :rtype: List[Interval]
12. """
13. **if** **not** intervals:
14. **return** []
15. res=[]
16. intervals.sort(key=**lambda** x:x.start)
17. temp=intervals[0]
18. **for** i **in** range(len(intervals)-1):
19. **if** temp.end<intervals[i+1].start:
20. res.append(temp)
21. temp=intervals[i+1]
22. **else**:
23. temp.end=max(intervals[i+1].end,temp.end)
24. res.append(temp)
25. **return** res

75颜色分类

题目：给定一个包含红色、白色和蓝色，一共 *n* 个元素的数组，**[原地](https://baike.baidu.com/item/%E5%8E%9F%E5%9C%B0%E7%AE%97%E6%B3%95" \t "_blank)**对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。

此题中，我们使用整数 0、 1 和 2 分别表示红色、白色和蓝色。

**注意:**  
不能使用代码库中的排序函数来解决这道题。

**示例:**

**输入:** [2,0,2,1,1,0]

**输出:** [0,0,1,1,2,2]

**进阶：**

* 一个直观的解决方案是使用计数排序的两趟扫描算法。  
  首先，迭代计算出0、1 和 2 元素的个数，然后按照0、1、2的排序，重写当前数组。
* 你能想出一个仅使用常数空间的一趟扫描算法吗？

思路：设置两个头尾指针,头指针pHead指向的位置是0该放置的位置,尾指针pTail指向的位置是2该放置的位置。i用来遍历整个数组,碰到0把它和pHead指向的数交换,碰到2把它和pTail指向的数交换,碰到1继续向后遍历。有点类似快速排序的分割数组这一步。

代码：

1. :type nums: List[int]
2. :rtype: void Do **not** **return** anything, modify nums **in**-place instead.
3. """
4. **if** nums == []:
5. **return**
6. pHead = 0
7. pTail = len(nums) - 1
8. i = 0
9. **while** i <= pTail :
10. **if** nums[i] == 2:
11. nums[i], nums[pTail ] = nums[pTail], nums[i]
12. pTail  -= 1
13. **elif** nums[i] == 0:
14. nums[i], nums[pHead] =nums[pHead], nums[i]
15. pHead += 1
16. i += 1
17. **else**:
18. i += 1

147对链表进行插入排序

148排序链表

在链表题目中写过了

179最大数

题目：给定一组非负整数，重新排列它们的顺序使之组成一个最大的整数。

**示例 1:**

**输入:** [10,2]

**输出:** 210

**示例 2:**

**输入:** [3,30,34,5,9]

**输出:** 9534330

**说明:** 输出结果可能非常大，所以你需要返回一个字符串而不是整数

思路：按照顺序拼接两个字母串进行比较，如果a ＋b串 大于 b＋a串，那么a比较大（即题意中理解的a应该放在前面），反之b比较大。

代码：

1. **class** Solution(object):
2. **def** largestNumber(self, nums):
3. """
4. :type nums: List[int]
5. :rtype: str
6. """
7. comp=**lambda** a,b:1 **if** a+b<b+a **else** -1
8. nums=map(str,nums)
9. nums.sort(cmp=comp)
10. **return** str(int("".join(nums)))

274H指数

题目：给定一位研究者论文被引用次数的数组（被引用次数是非负整数）。编写一个方法，计算出研究者的 h 指数。

[h 指数的定义](https://baike.baidu.com/item/h-index/3991452?fr=aladdin): “一位有 h 指数的学者，代表他（她）的 N 篇论文中至多有 h 篇论文，分别被引用了至少 h 次，其余的 N - h 篇论文每篇被引用次数不多于 h 次。”

示例:

输入: citations = [3,0,6,1,5]

输出: 3

解释: 给定数组表示研究者总共有 5 篇论文，每篇论文相应的被引用了 3, 0, 6, 1, 5 次。

  由于研究者有 3 篇论文每篇至少被引用了 3 次，其余两篇论文每篇被引用不多于 3 次，所以她的 h 指数是 3。

说明: 如果 h 有多种可能的值，h 指数是其中最大的那个。

思路：

代码：

1. **class** Solution(object):
2. **def** hIndex(self, citations):
3. """
4. :type citations: List[int]
5. :rtype: int
6. """
7. citations.sort(reverse=False)
8. h = len(citations)
9. index = 0
10. **while** index < len(citations):
11. **if** citations[index] >= h:
12. **return** h
13. index += 1
14. h -= 1
15. **return** 0

324摆动排序II

题目：给定一个无序的数组 nums，将它重新排列成 nums[0] < nums[1] > nums[2] < nums[3]... 的顺序。

**示例 1:**

**输入:** nums = [1, 5, 1, 1, 6, 4]

**输出:** 一个可能的答案是 [1, 4, 1, 5, 1, 6]

**示例 2:**

**输入:** nums = [1, 3, 2, 2, 3, 1]

**输出:** 一个可能的答案是 [2, 3, 1, 3, 1, 2]

**说明:**  
你可以假设所有输入都会得到有效的结果。

**进阶:**  
你能用 O(n) 时间复杂度和 / 或原地 O(1) 额外空间来实现吗？

思路：排序后，奇偶数分开插入

代码：

1. **class** Solution(object):
2. **def** wiggleSort(self, nums):
3. """
4. :type nums: List[int]
5. :rtype: void Do not return anything, modify nums in-place instead.
6. """
7. **if** **not** nums:
8. **return** []
9. tmp = sorted(nums)
10. **for** i **in** range(1, len(nums), 2):
11. nums[i] = tmp.pop()
12. **for** i **in** range(0, len(nums), 2):
13. nums[i] = tmp.pop()

524通过删除字母匹配到字典里最长单词

题目：给定一个字符串和一个字符串字典，找到字典里面最长的字符串，该字符串可以通过删除给定字符串的某些字符来得到。如果答案不止一个，返回长度最长且字典顺序最小的字符串。如果答案不存在，则返回空字符串。

**示例 1:**

**输入:**

s = "abpcplea", d = ["ale","apple","monkey","plea"]

**输出:**

"apple"

**示例 2:**

**输入:**

s = "abpcplea", d = ["a","b","c"]

**输出:**

"a"

**说明:**

1. 所有输入的字符串只包含小写字母。
2. 字典的大小不会超过 1000。
3. 所有输入的字符串长度不会超过 1000。

思路：首先对字典按照输出要求进行排序，即按照长度逆序，对于长度相同的按照字典顺序小的排列，这样当我们找到第一个字符串以后就不需要继续匹配字符串了。匹配字符串的时候使用两个下标分别指向字符串和字典中的字符串

代码：

1. **class** Solution(object):
2. **def** findLongestWord(self, s, d):
3. """
4. :type s: str
5. :type d: List[str]
6. :rtype: str
7. """
8. d.reverse()
9. res = []
10. maxlen = 0
11. flag = 0
12. d = sorted(d, key = **lambda** x :len(x))
13. **for** item **in** d[::-1]:
14. **if** flag == 1 **and** len(item) < maxlen:
15. **break**
16. i = j = 0
17. **while** i < len(item) **and** j < len(s):
18. **if** item[i] == s[j]:
19. i += 1
20. j += 1
21. **if** i == len(item):
22. flag = 1
23. maxlen = len(item)
24. res.append(item)
25. **if** res == []:
26. **return** ""
27. **elif** len(res) == 1:
28. **return** res[0]
29. res.sort()
30. **return** res[0]

767重构字符串（难）

题目：给定一个字符串S，检查是否能重新排布其中的字母，使得两相邻的字符不同。

若可行，输出任意可行的结果。若不可行，返回空字符串。

**示例 1:**

**输入:** S = "aab"

**输出:** "aba"

**示例 2:**

**输入:** S = "aaab"

**输出:** ""

**注意:**

* S 只包含小写字母并且长度在[1, 500]区间内。

思路：统计这个字符串每个字符的出现频率，在组成新字符串时，在满足和上一个字符不相同的情况下，优先使用最流行的字符串。每用一个字符都对counter进行更新，直至循环结束。注意如果次数为0要删除，否则counter不会为空

代码：

1. counter = collections.Counter(S)
2. ans = "#"
3. **while** counter:
4. stop = True
5. **for** item, times **in** counter.most\_common():
6. **if** ans[-1] != item:
7. ans += item
8. counter[item] -= 1
9. **if** **not** counter[item]:
10. **del** counter[item]
11. stop = False
12. **break**
13. **if** stop: **break**
14. **return** ans[1:] **if** len(ans) == len(S) + 1 **else** ""